Exhibit B

Java Persistence Framework

- Design Decisions
- How to Use It

Best Available Copy

Design - Domain Objects

- Domain Objects - BITS CRM View
  - Domain objects need to be pure in implementation.
  - Need to be independent of any architectural layer including persistence.
  - Need database neutrality and abstraction.

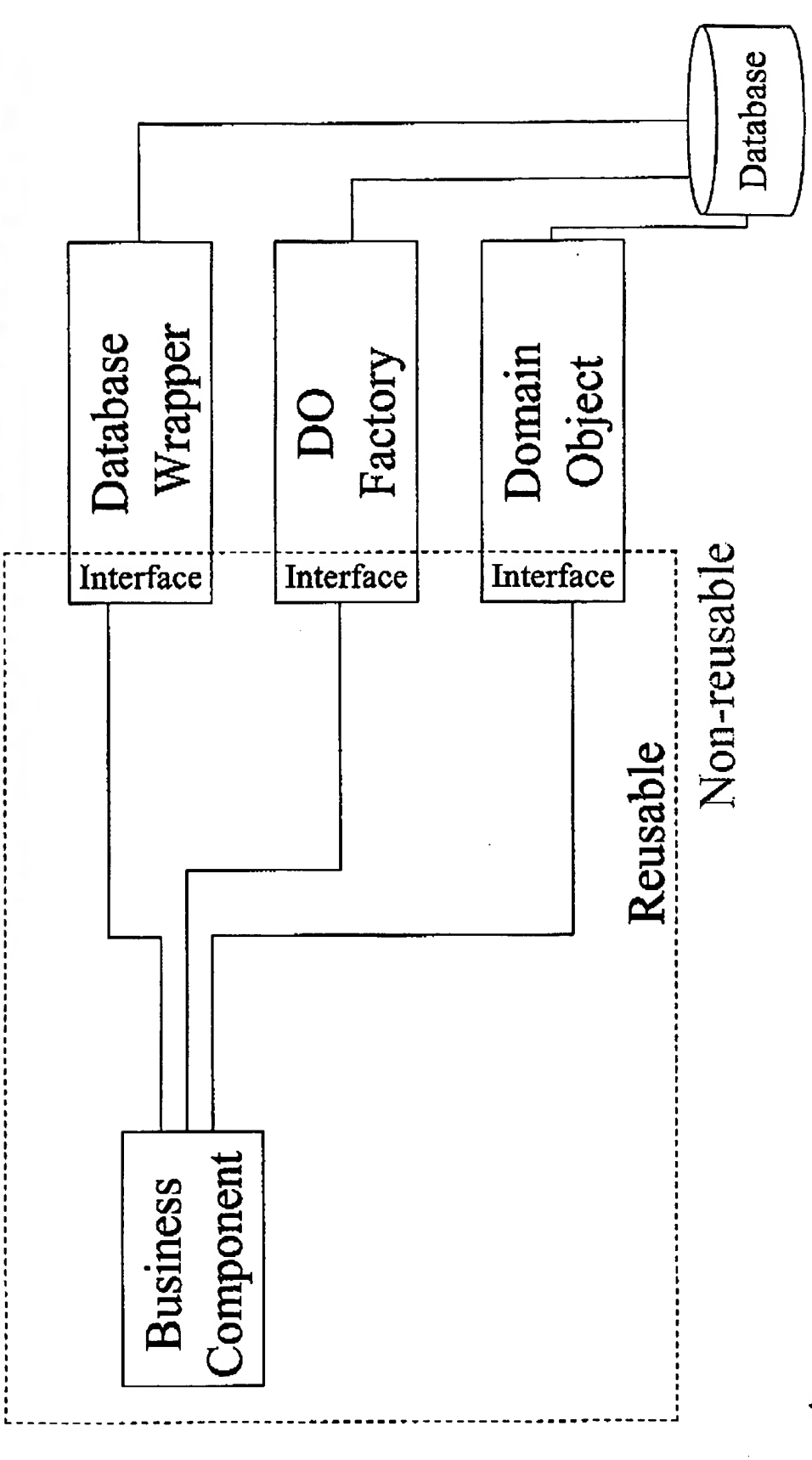
Design - Domain Objects

- However, domain objects themselves have to be tied to the database in which they are stored due to the differences between database vendors.
- Therefore, we create a **Domain Object Interface** which is wholly reusable and database neutral.
- This interface for a specific Domain Object is all that the Business Component sees.
- An *implementation* of a Domain Object must be done for each database.
- With this approach, the Business Component remains independent from the implementation.

Design - Business Components

- Business Components
  - Contain the business rules or functionality of the application.
  - Should be wholly reusable and independent of data source.
  - Deals with a finite set of Domain Objects
  - Implemented as Session EJB's.

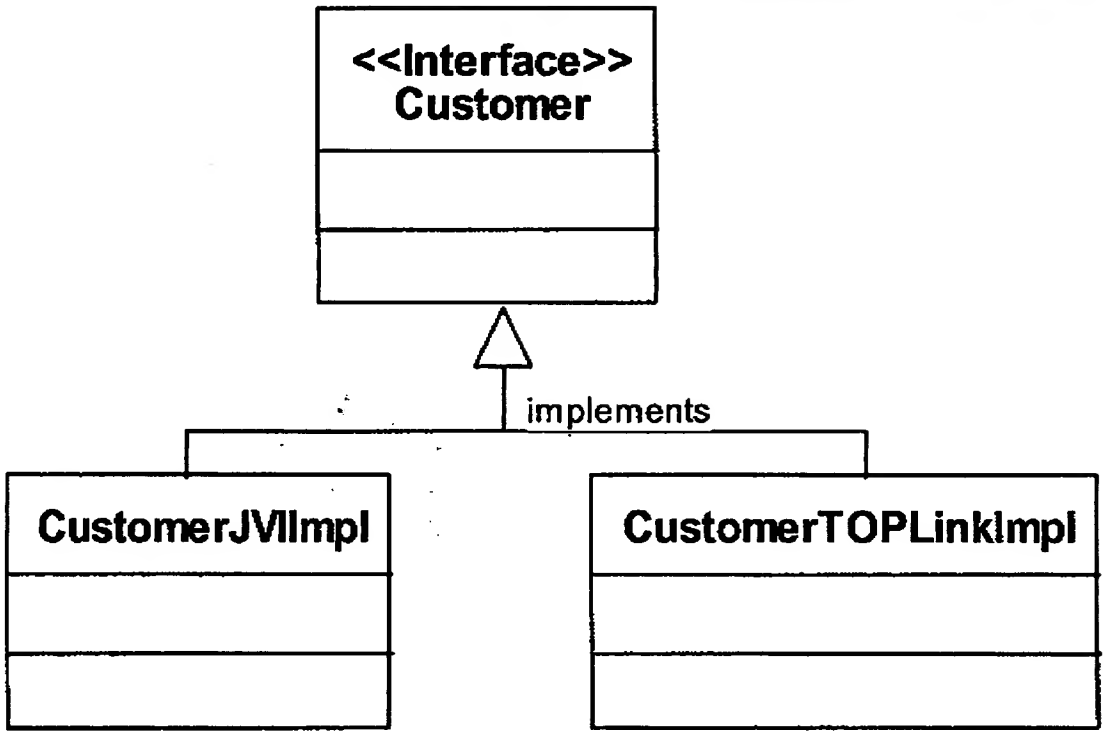
Design - Reuse

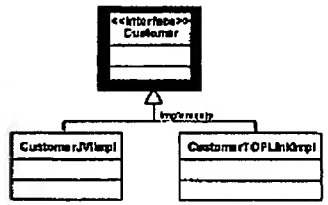


Sprint Restricted

Sprint

Domain Object - Class Diagram



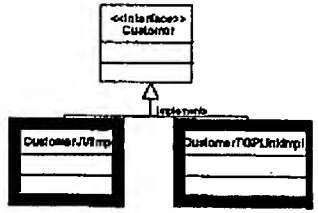


Domain Object - Developer's Guide

Interface

- Keep business components and other code neutral to the implementation of domain objects.
- Should define any behavior for the domain object as well as accessors.
- Example:

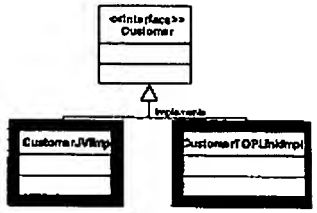
```
public interface Customer {
  //Accessors for name
  public String getName();
  public void setName(String name);

  //Accessors for address
  public Address getAddress();
  public void setAddress(Address address);

  public boolean isValid();
}
```



Domain Object - Developer's Guide

- Implementation
  - A class that implements the DO interface.
  - DO classes are named to reflect the database in which they reside.
  - Example:

```
public class CustomerTOPLinkImpl implements Customer {
        private String name;
        private Handle address;

        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public Address getAddress() {
                //return my address field, implemented later
        }
        public void setAddress(Address address) {
                //set my address field, implemented later
        }
}
```



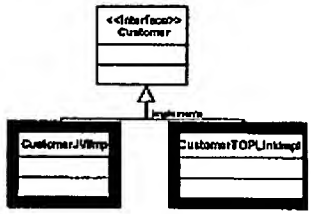
Domain Object - Developer's Guide

- Implementation (continued)
  - A business component has visibility to the domain object interface type, not the class type. So, the following would ***NOT*** be correct:

```
public class BusinessComponent
  public Customer createCustomer(args) {
        Customer customer = new CustomerTOPLinkImpl();
        return customer;
  }
}
```

  - The problem with this code is that by calling **new** on the class **CustomerTOPLinkImpl**, the Business Component must be re-implemented if the database for Customer domain objects changes.
  - Business Components must never use **new** directly. Using a factory is the correct method.

Domain Object - Developer's Guide



Accessors

- Example:

```
public class CustomerTOPLinkImpl implements Customer {
  private ValueHolderInterface m_address;

  //return type Address is an interface
  public Address getAddress() {
          return (Address)m_address.getValue();
  }
  public void setAddress(Address address) {
          if (m_address == null)
                  m_address = new ValueHolder();
          m_address.setValue(address);
  }
  public ValueHolderInterface getAddressHolder() {
          return m_address;
  }
  public void setAddressHolder(ValueHolderInterface address) {
          m_address = address;
  }
```

Extracts the Address object from the ValueHolder

Transient DO - Class Diagram

CustomerTransientImpl
Non-Database field types.
Constructors and the createTransient() method.

Transient DO - Developer's Guide

- Typically, business component code follows the following steps:
  - Obtain a database wrapper
  - Begin a session
  - Obtain factories
  - Create/Find domain objects
  - Commit transaction
  - End the session
  - Return the request information to the application controller
- The returned information *can* be in the form of domain objects.

Transient DO - Developer's Guide

- Domain objects are considered **persistent** while in the context of a database session and are not valid outside of the database session.
- When a business component wants to pass back a domain object, the object must be made transient.
- To accomplish this, the developer must add a method to the domain object and create a transient class.

Transient DO - Developer's Guide

- A conversion method, **createTransient()**, is defined on each domain object interface and be implemented by the DO implementation.
- Example:

```
//Database implementation
public class CustomerTOPLinkImpl implements Customer {
  private String m_name;
  private ValueHolderInterface m_address;

  Customer createTransient() {
        return new CustomerTransientImpl(name,

      getAddress().createTransient());
  }
}
```

Transient DO - Developer's Guide

- A **CustomerTransientImpl** class is created. The CustomerTransientImpl contains no database specific types or codes.
- Example:

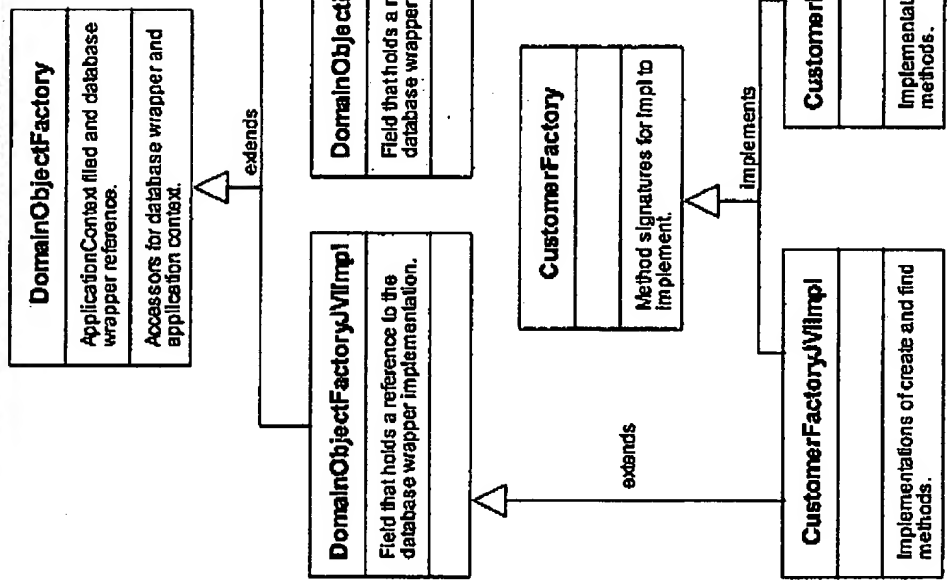
```
//Transient implementation
public class CustomerTransientImpl implements Customer {
   private String name;
   private Address address;

   public CustomerTransientImpl(String name, Address address) {
       this.name = name;  this.address = address;
   }
   Customer createTransient() {
       return this;
   }
}
```

Transient DO - Developer's Guide

⌘ A Business Component might look like the following:

```
public Customer getCustomer(String id) {
   // Get the database wrapper from the DatabaseWrapper class
   DatabaseWrapper dbWrapper=DatabaseWrapper.GetDatabaseWrapper(aProperties);

   // begin a session
   dbWrapper.beginSession();

   // get the customer factory
   CustomerFactory cf =
       (CustomerFactory)dbWrapper.getDomainObjectFactory(CustomerFactory.class);

   // get a customer from the factory using a find method
   Customer customer = cf.findById(id);
... ...
  // while still in the session, convert the persistent customer to transient
   Customer transientCustomer = customer.transientCopy();

   // end the session
   DbWrapper.endSession();

   // return the transient customer
   return transientCustomer;
}
```

Factories - Class Diagram



Sprint.

Sprint Restricted

DO Factory - Developer's Guide

- The factory framework parallels the domain object framework. The developer implements a factory interface for each database containing its domain objects.
- Domain object factories create and retrieve domain objects from the database.
- Business components get a reference to a specific domain object factory and ask it to create or find the domain object for it.
- Factory methods return domain object interface types.

DO Factory - Developer's Guide

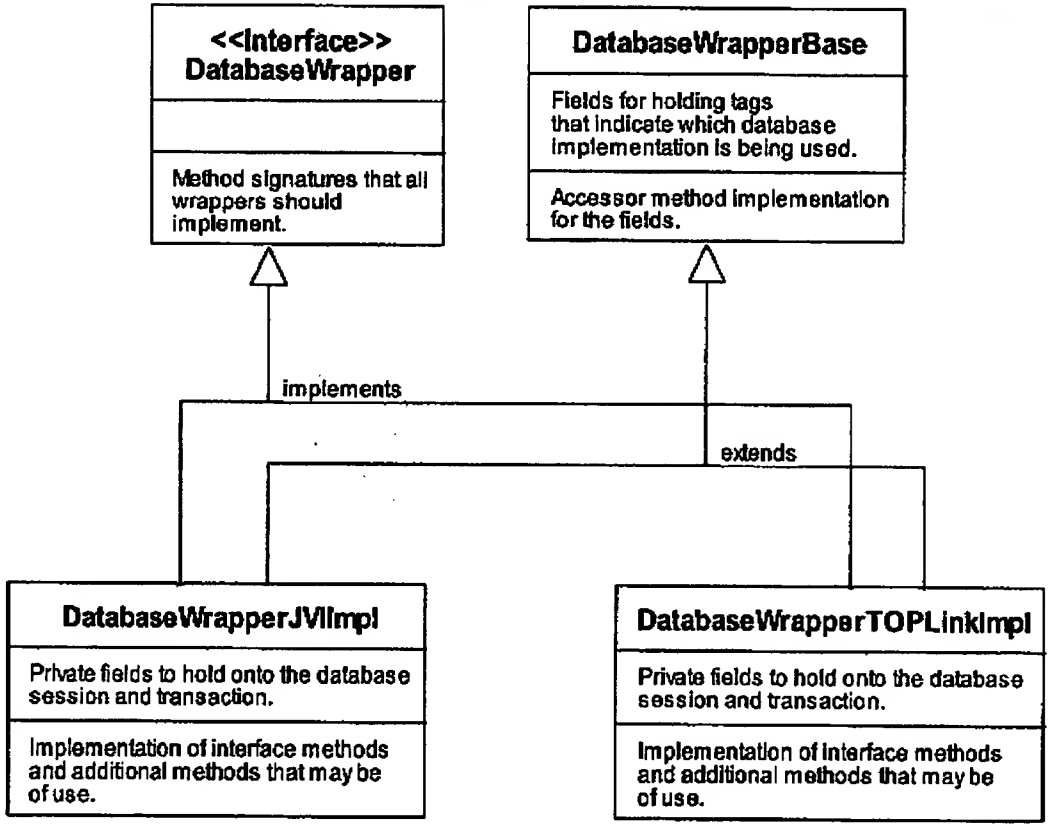
- Example Interface and Implementation:

```
public interface CustomerFactory {
   public Customer create();
   public Customer findByName(String name);
   public Vector findAll();
}
public class CustomerFactoryJVIImpl extends DomainObjectFactoryJVIImpl
                                    implements CustomerFactory {
   public Customer create() {
         return new CustomerJVIImpl();
   }
   public Customer findByName(String name) {
         //code to query versant for a Customer by name
         return customer;
   }
   public Vector findAll() {
         //code to query versant for all customers
         return customers;
   }
}
```

DO Factory - Developer's Guide

- Any query that needs to be performed for a given domain object type is placed in a "find" method on the domain object factory.
- The find method takes full advantage of database specific code to generate queries.
- Since a factory implementation is specific to a given database, there must be a factory for the domain object factories. This is also parallel to the domain object framework.
- A business component cannot directly instantiate a domain object factory. The DatabaseWrapper is used for this.

Database Wrapper - Class Diagram



DB Wrapper - Developer's Guide

- The database wrapper provides an API for session and transaction management.
- It is also the factory for domain object factories.
- The developer simply calls a method on the wrapper specifying the class of the domain object factory to receive. The return type is Object, so the developer must cast the return value to the specific factory type.

```
DatabaseWrapper dbWrapper =
   DatabaseWrapperBase.GetDatabaseWrapper(aProperties);
CustomerFactory customerFactory =
   (CustomerFactory)dbWrapper.getDomainObjectFactory(CustomerFactory.class);
```

Database Wrapper - Developer's Guide

- GetDatabaseWrapper()
  - Takes aProperties object that contains all the necessary properties to instantiate aDatabaseWrapper, including the logical datasource name, as its parameter and returns an appropriate DatabaseWrapper object.
  - The ini file for the application represented by the Properties object contains a mapping of datasources to database wrapper types.

EJB Manager - Developer's Guide

- GetEJBHome()
  - Takes an Properties object and a class object for the home interface and returns remote interface object of the EJB.
  - The ini file from which the Properties is derived contains a mapping of the home interfaces to the name registered with the Naming Services (ie. JNDI).
  - The result must be cast back to the correct remote interface type, then call create() on it to obtain the remote reference to the EJB.

Summary

- Developer Tasks
  - Create the domain object interface
  - Create the database-specific implementation class
  - Create the transient domain object implementation
  - Create the factory interface
  - Create the factory implementation

Resources

- ION Tech Arch Website
  - http://pinpoint/bits/bta/ion_ta/
  - SIDM 2.0 Technical Reference
    - Information Section
- BTA Self Paced Training Website
  - http://pinpoint/bits/bta/training/
- "Using TOPLink Builder"
- "Java Versant Interface Overview"